Docket No. AUS920030792US1 *PATENT*

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: **Best et al.**	§	
	§	Group Art Unit: **2163**
Serial No. **10/697,899**	§	
	§	Examiner: **Ho, Binh Van**
Filed: **October 30, 2003**	§	
	§	
For: **Method and Apparatus for Increasing Efficiency of Data Storage in a File System**	§	

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

35525
PATENT TRADEMARK OFFICE
CUSTOMER NUMBER

APPEAL BRIEF (37 C.F.R. 41.37)

This brief is in furtherance of the Notice of Appeal, filed in this case on January 17, 2007.

A fee of $500.00 is required for filing an Appeal Brief. Please charge this fee to IBM Corporation Deposit Account No. 09-0447. No additional fees are believed to be necessary. If, however, any additional fees are required, I authorize the Commissioner to charge these fees which may be required to IBM Corporation Deposit Account No. 09-0447. No extension of time is believed to be necessary. If, however, an extension of time is required, the extension is requested, and I authorize the Commissioner to charge any fees for this extension to IBM Corporation Deposit Account No. 09-0447.

REAL PARTY IN INTEREST

The real party in interest in this appeal is the following party: International Business Machines Corporation of Armonk, New York.

RELATED APPEALS AND INTERFERENCES

With respect to other appeals or interferences that will directly affect, or be directly affected by, or have a bearing on the Board's decision in the pending appeal, there are no such appeals or interferences.

STATUS OF CLAIMS

A. TOTAL NUMBER OF CLAIMS IN APPLICATION

Claims in the application are: 1-22.

B. STATUS OF ALL THE CLAIMS IN APPLICATION

1. Claims canceled: None.
2. Claims withdrawn from consideration but not canceled: None.
3. Claims pending: 1-22.
4. Claims allowed: None.
5. Claims rejected: 1-22.
6. Claims objected to: None.

C. CLAIMS ON APPEAL

The Claims on appeal are: 1-22.

STATUS OF AMENDMENTS

No amendments were presented after the Final Office Action of October 18, 2006.

SUMMARY OF CLAIMED SUBJECT MATTER

A. CLAIM 1 - INDEPENDENT

The subject matter of claim 1 is directed to a method in a data processing system for storing data in a file system (Specification, p. 8, ll. 16-18). The method includes determining whether space is available in an inode for a file in the file system (Specification, p. 8, ll. 16-22; p. 10, l. 26 through l. 14; p. 14, ll. 10-11; Figure 6, reference numeral 600). The method further includes, responsive to space being available, storing the data in the inode (Specification, p. 8, ll. 16-22; p. 10, l. 26 through 1. 14; p. 14, ll. 1-4; Figure 5, reference numeral 508; Figure 6, reference numeral 604).

B. CLAIM 8 - INDEPENDENT

The subject matter of claim 8 is directed to a data processing system for storing data in a file system (Specification, p. 8, ll. 16-18). The data processing system includes a bus system (Figure 2, reference numeral 206), a communications unit connected to the bus system (Figure 2, reference numerals 210, 212, and 222), and a memory connected to the bus system (Figure 2, reference numerals 208 and 204), wherein the memory includes a set of instructions (Specification, p. 7, ll. 5-9). The data processing system further includes a processing unit connected to the bus system (Figure 2, reference numeral 202). The processing unit executes the set of instructions to determine whether space is available in an inode of the file in the file system (Specification, p. 8, ll. 16-22; p. 10, l. 26 through l. 14; p. 14, ll. 10-11; Figure 6, reference numeral 600). The processing unit also executes the instruction to store the data in the inode in response to space being available (Specification, p. 8, ll. 16-22; p. 10, l. 26 through 1. 14; p. 14, ll. 1-4; Figure 5, reference numeral 508; Figure 6, reference numeral 604).

C. CLAIM 9 - INDEPENDENT

The subject matter of claim 9 is directed to a data processing system for storing data in a file system (Specification, p. 8, ll. 16-18). The data processing system includes determining means (Figure 2, reference numeral 202) for determining whether space is available in an inode of the file in the file system (Specification, p. 8, ll. 16-22; p. 10, l. 26 through l. 14; p. 14, ll. 10-11; Figure 6,

reference numeral 600). The data processing system further includes storing means (Figure 2, reference numerals 226, 228, and 230), responsive to space being available, for storing the data in the inode (Specification, p. 8, ll. 16-22; p. 10, l. 26 through 1. 14; p. 14, ll. 1-4; Figure 5, reference numeral 508; Figure 6, reference numeral 604).

D. CLAIM 10 - DEPENDENT

The subject matter of claim 10 is directed to the data processing system of claim 9. The determining means is a first determining means (Figure 2, reference numeral 202) and the storing means is a first storing means (Figure 2, reference numerals 226, 228, and 230). The subject matter of claim 10 further includes second determining means (Figure 2, reference numeral 202) for determining whether additional data is present (Specification, p. 12, ll. 2-15), and second storing means (Figure 2, reference numerals 226, 228, and 230), responsive to the additional data being present, for storing the additional data in a partially filled block of another file (Specification, p. 12, ll. 2-15).

E. CLAIM 11 - DEPENDENT

The subject matter of claim 11 is directed to the data processing system of claim 9. The storing means is a first storing means (Figure 2, reference numerals 226, 228, and 230). The subject matter of claim 11 further includes second storing means (Figure 2, reference numerals 226, 228, and 230), responsive to spacing being unavailable, for storing the additional data in a partially filled block of another file (Specification, p. 12, ll. 2-15).

F. CLAIM 14 - DEPENDENT

The subject matter of claim 14 is directed to the data processing system of claim 9. The determining means is a first determining means (Figure 2, reference numeral 202) and the storing means is a first storing means (Figure 2, reference numerals 226, 228, and 230). The subject matter of claim 14 further includes second determining means (Figure 2, reference numeral 202) for determining whether a file size for the data is divisible by a block size for blocks in the file system (Specification, p. 12, l. 16 through p. 13, l. 7; p. 14, ll. 14-16; p. 15, ll. 17-13; Figure 6, reference numeral 602; Figure 7, reference numeral 700). The subject matter of claim 14 further includes

second storing means (Figure 2, reference numerals 226, 228, and 230), if the file size is divisible the by the block size, for storing the data in a block (Specification, p. 14, ll. 24-29; p. 15, ll. 7-12; Figure 6, reference numeral 604; Figure7, reference numerals 702 and 716) .

G. CLAIM 15 - DEPENDENT

The subject matter of claim 15 is directed to the data processing system of claim 9. The determining means is a first determining means (Figure 2, reference numeral 202) and the storing means is a first storing means (Figure 2, reference numerals 226, 228, and 230). The subject matter of claim 15 further includes second determining means (Figure 2, reference numeral 202) for determining whether space is available in the inode to store the data (Specification, p. 8, ll. 16-22; p. 10, l. 26 through l. 14; p. 14, ll. 10-11; Figure 6, reference numeral 600). The subject matter of claim 15 further includes second storing means (Figure 2, reference numerals 226, 228, and 230), responsive to room being unavailable in the inode, for storing the data in a partially filled block of another file (Specification, p. 12, ll. 2-15).

H. CLAIM 16 - INDEPENDENT

The subject matter of claim 16 is directed to a computer program product in a computer readable medium for storing data in a file system (Specification, p. 8, ll. 16-18; p. 16, l. 24 through p. 17, l. 12). The computer program product includes first instructions for determining whether space is available in an inode of the file in the file system (Specification, p. 8, ll. 16-22; p. 10, l. 26 through l. 14; p. 14, ll. 10-11; Figure 6, reference numeral 600). The computer program product further includes second instructions, responsive to space being available, for storing the data in the inode (Specification, p. 8, ll. 16-22; p. 10, l. 26 through 1. 14; p. 14, ll. 1-4; Figure 5, reference numeral 508; Figure 6, reference numeral 604).

GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

The grounds of rejection to review on appeal are as follows:

A. GROUND OF REJECTION 1 (Claims 1-22)

Whether *Crow et al.*, Versatile Indirection in an Extent Based File System, U.S. Patent Application Publication 2004/0254907 (December 16, 2004) (hereinafter "*Crow*") fails to anticipate claims 1-22.

ARGUMENT

A. GROUND OF REJECTION 1 (Claims 1-22)

A.1. Claims 1, 8, 9, and 16

A.1.i. *Response to Rejection*

Claim 1 is a representative claim of this grouping of claims. Claim 1 is as follows:

> 1. A method in a data processing system for storing data in a file system, the method comprising:
>
> determining whether space is available in an inode for a file in the file system; and
>
> responsive to space being available, storing the data in the inode.

The Examiner rejects claim 1 as anticipated by *Crow*. In regards to claim 1, the Examiner asserts that:

> *Crow* discloses in figure 8C, a method in a data processing system for storing data in a file system, the method comprising determining whether space is available in an inode for a file in the file system; and responsive to space being available, storing the data in the inode (paragraph [0048], [0053]).

Final Office Action dated October 18, 2006, p. 2.

A prior art reference anticipates the claimed invention under 35 U.S.C. §102 only if every element of a claimed invention is identically shown in that single reference, arranged as they are in the claims. *In re Bond*, 910 F.2d 831, 832, 15 U.S.P.Q.2d 1566, 1567 (Fed. Cir. 1990). All limitations of the claimed invention must be considered when determining patentability. *In re Lowry*, 32 F.3d 1579, 1582, 32 U.S.P.Q.2d 1031, 1034 (Fed. Cir. 1994). Anticipation focuses on whether a claim reads on the product or process a prior art reference discloses, not on what the reference broadly teaches. *Kalman v. Kimberly-Clark Corp.*, 713 F.2d 760, 218 U.S.P.Q. 781 (Fed. Cir. 1983). In this case each and every feature of the presently claimed invention is not identically shown in the cited reference, arranged as they are in the claims.

Crow does not anticipate claim 1 because *Crow* does not teach all the features of claim 1. Claim 1 is as follows:

> 1. A method in a data processing system for storing data in a file system, the method comprising:
>
> determining whether space is available in an inode for a file in the file system; and
>
> responsive to space being available, storing the data in the inode.

Specifically, *Crow* does not teach the feature of, "determining whether space is available in an inode *for a file in the file system* and responsive to space being available, storing the *data* in the inode" as recited in claim 1. The Examiner asserts otherwise, citing *Crow*'s figure 8C, reproduced below:

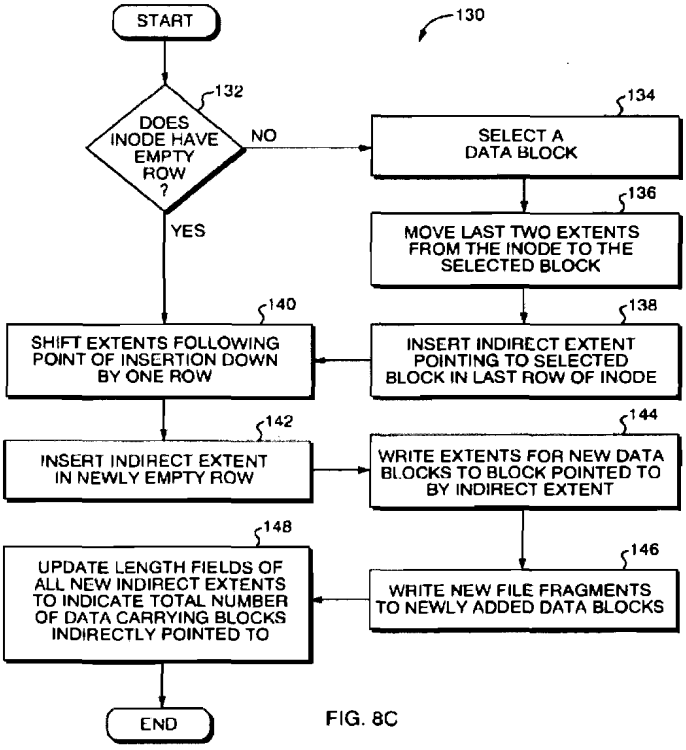

FIG. 8C

According to figure 8C, in step 132, "the operating system first determines whether at least one empty row remains for writing a new *extent* to the file's inode." *Crow*, paragraph 41 (emphasis added). In step 146, the new *file fragments are stored in newly added data block* and not in the inode. The inode stores the extent which contains a pointer that indicates both the logical volume and a physical offset of the data block. *Crow*, paragraphs 33-34. Therefore, figure 8C does not teach the features of claim 1 because figure 8C determines whether there is space in the inode

for an "extent" and not for a file *in the file system* as recited in claim 1. Furthermore, figure 8C discloses storing the data in a newly added data block instead of storing the data in the inode as recited in claim 1.

Additionally, the Examiner cites to the following portions of *Crow* as disclosing the features of claim 1:

> [0048] Later, a request from a software application for more *data blocks for the file* is received by the operating system (step 157). In response to the request, the operating system determines whether the region contiguous to the physical location of the previous segment of the file has more available data blocks (step 158). If region has more available blocks, the operating system allocates a new string of blocks immediately following the physical location previous segment, i.e., contiguous with the previous segment (step 160). Then, the operating system increases the value of the length stored in the length field of the previous extent for the region by the number of blocks in the new string (step 161). If no blocks contiguous to the previous segment are available, the operating system again searches for a logical volume with a larger than average contiguous region of available data blocks (step 162). The newly found logical volume may be a different logical volume. Thus, the new string of data blocks may be allocated to the file from a different logical volume.
>
> [0053] The operating system writes the binary value to the third entry 176 to indicate storage of a data file when the associated inode is first created. Then, the operating system uses the inode to store the associated data file. When the size of the data file surpasses the limited space available in the inode, the operating system converts the inode to an inode for storage of lists of extents.

Crow, paragraphs 48 and 53.

Neither the cited portion nor any other portion of *Crow* teaches the feature of, "determining whether space is available in an inode *for a file in the file system* and *responsive* to space being available, storing the *data* in the inode," as recited in claim 1. Paragraph 48 only discusses an application requesting more data blocks for a file. If there are more data blocks contiguous to the physical location of the previous segment of the file, then the operating system allocates a new string of blocks immediately following the physical location of the previous segment. If no blocks contiguous to the previous segment are available, the operating system again searches for a logical volume with a larger than average contiguous region of available data blocks (step 162).

However, nothing in paragraph 48 discloses the features of claim 1. Therefore paragraph 48 is completely irrelevant to the claimed feature, "determining whether space is available in an inode *for a file in the file system* and *responsive* to space being available, storing the *data* in the inode," as recited in claim 1.

Similarly, paragraph 53 discloses the use of an inode to store a data file. However, nothing in paragraph 53 or any other portion of *Crow* teaches the precondition to storing the data in the in the inode as recited in claim 1. In other words, *Crow* does not teach, "*determining* whether space is available in an inode *for a file in the file system* and *responsive* to space being available," as recited in claim 1. *Crow* does not perform a pre-determination prior to storing the data in the inode. *Crow* performs a post-determination. *Crow* performs the function of storing the data in the inode and "when the size of the data file surpasses the limited space available in the inode, the operating system converts the inode to an inode for storage of lists of extents." *Crow*, paragraph 53. Therefore, this portion of *Crow* does not teach, "*determining* whether space is available in an inode *for a file in the file system* and *responsive* to space being available, storing the data in the inode," as recited in claim 1.

As shown above, *Crow* does not teach all of the features of claim1. Accordingly, *Crow* does not anticipate claim 1 or any other claim in this grouping of claims.

A.1.ii. *Rebuttal of the Examiner's Response*

In response, *vis-à-vis* claim 1, the Examiner states that:

> The Examiner respectfully disagreed with the Applicant's argument above; since Crow discloses (paragraph [0044], [0047] and [0048]) "the operating system determines the maximum number of available contiguous blocks in each logical volume from data in the volume's header or from information in a superblock spanning the entire storage system."

Final Office Action of October 18, 2006, p. 5.

The Examiner makes a single counter argument based on the above-cited quotation. The Examiner does not address any of the specific points raised by Applicants.

However, the one quotation cited by the Examiner deals with the asserted feature in *Crow* that the operating system determines the maximum number of available contiguous blocks in each logical volume from data in the volume's header, or from information in a superblock. However, this quotation has utterly nothing to do with the claimed feature.

According to Applicant's specification, p. 8, ll. 24-25, an inode is a data structure or record used to store information *about* files (emphasis supplied). This definition comports with what one of ordinary skill in the art knows an inode to be. Even if true, the fact that the operating system determines the maximum number of available contiguous blocks in each logical volume or from information in a superblock has little to do with the claimed feature of, "determining whether space is available in an inode for a file in the file system," as in claim 1. Even if a superblock were somehow the equivalent of an inode, *Crow* would then only be teaching that the superblock contained information *about* files. However, *Crow* does not teach *determining* whether space is available *in* the inode for a file in the file system. Therefore, the Examiner has failed to rebut the fact that *Crow* does not teach all of the features of claim 1. Accordingly, *Crow* does not anticipate claim 1 or any other claim in this grouping of claims.

A.2. Claims 2, 3, 7, 10, 11, 15, 17, 18, and 22

A.2.i. *Response to Rejection*

Claim 2 is a representative claim of this grouping of claims. Claim 2 is as follows:

> 2. The method of claim 1 further comprising:
> determining whether additional data is present; and
> responsive to the additional data being present, storing the additional data in a partially filled block of another file.

The Examiner rejects claim 2 as anticipated by *Crow*. Regarding claim 2, the Examiner asserts:

> *Crow* discloses in figures 8A-8C1 and 10, to determining whether additional data being present; and responsive to the additional data being present, storing the additional data in a partially filled block of another file (paragraph [0038], [0039], [0042] and [0044]).

Final Office Action dated October 18, 2006, p. 2.

Crow does not anticipate claim 2 because *Crow* does not teach all the features of claim 2. Claim 2 is as follows:

> 2. The method of claim 1 further comprising:
> determining whether additional data is present; and
> responsive to the additional data being present, storing the additional data in a partially filled block of another file.

Because claim 2 depends from claim 1, the same distinctions between *Crow* and claim 1 apply to claim 2. Additionally, claim 2 claims other additional combinations of features not suggested by the reference. Specifically, *Crow* does not teach the feature of, "storing *the additional* data in a *partially filled* block *of another file,*" as recited in claim 2. The Examiner asserts otherwise, citing the following figures:

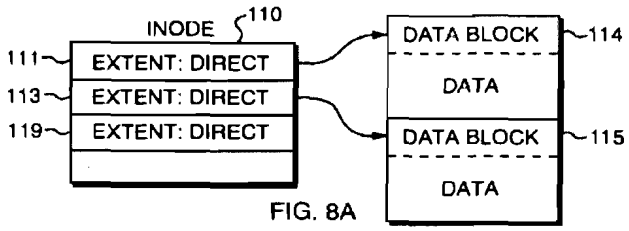

FIG. 8A

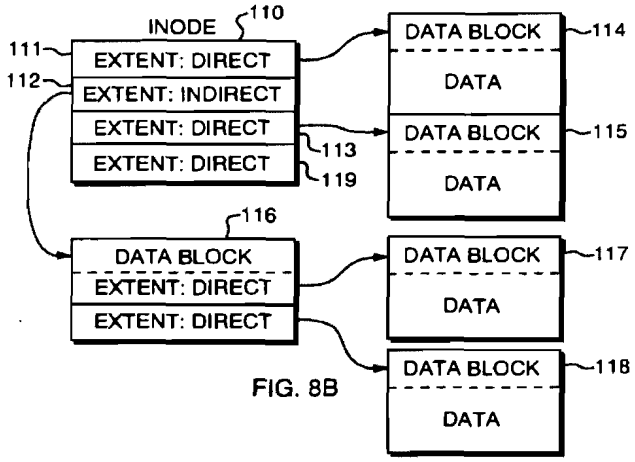

FIG. 8B

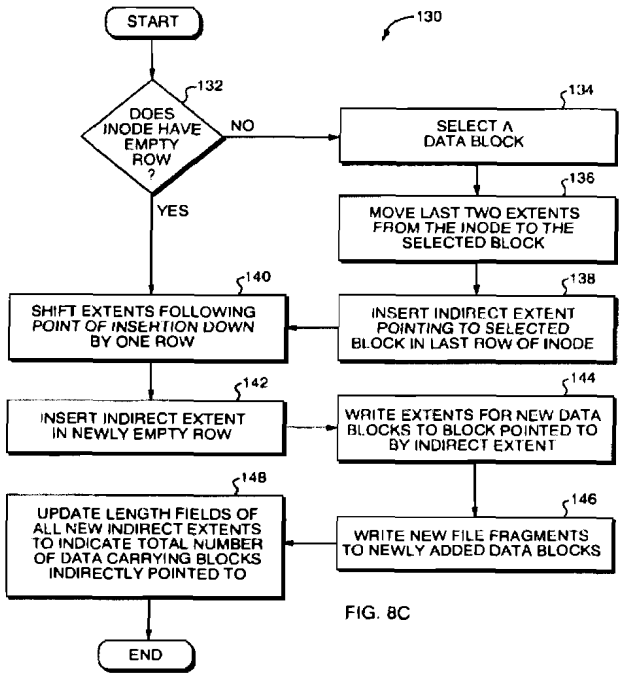

FIG. 8C

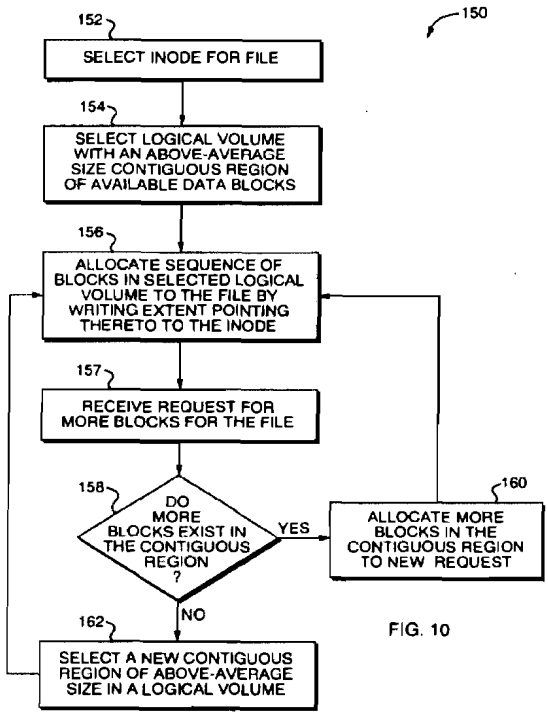

FIG. 10

Figures 8A, 8B, and 8C do not teach the feature of, "storing *the additional* data in a *partially* filled block *of another file.*" Instead, figures 8A and 8B show how the operating system uses indirect extents to grow the middle of a file. The indirect extent 112 points to more extents stored in a data block 116. These extents, in turn, point to ***new*** data block 117 and original data block 215. *Crow*, paragraph 38. This feature enables an operating system to logically insert a new data segment between any two selected data segments of a file without physically moving data blocks. *Crow*, paragraph 39. Figure 8C is a flow chart illustrating a method 130 of inserting a new file segment between two adjacent file segments. *Crow*, paragraph 39. In step 148, the operating system writes the new file segment in the ***new*** data block pointed to by the new direct extent. *Crow*, paragraph 44.

However, these figures do not teach "storing *the additional* data in a *partially filled* block *of another file,*" as recited in claim 2. Instead, these figures show how to add ***new*** data blocks to a file using *indirect extents*. Therefore, figures 8A, 8B, and 8C of *Crow* do not teach the claimed features of claim 2.

Furthermore, figure 10 of *Crow* does not teach the feature, "storing *the additional* data in a *partially filled* block *of another file.*" Figure 10 illustrates a method 150 of allocating data blocks

to a file from a plurality of logical volumes. The operating system allocates a string of data blocks from the contiguous region of the selected volume to the file by writing an extent. The extent points to the string in the first row of the inode assigned to the file (step 156). Step 157 occurs when an application requests more data blocks for a file. If there are more data blocks contiguous to the physical location of the previous segment of the file, then the operating system allocates a new string of blocks immediately following the physical location of the previous segment. If no blocks contiguous to the previous segment are available, the operating system searches for a logical volume with a larger than average contiguous region of available data blocks (step 162).

However, nothing in Figure 10 teaches the feature, "storing *the additional* data in a *partially filled* block *of another file.*" Instead, Figure 10 teaches a method of storing data in contiguous blocks, which is different from the features of claim 2. Therefore, figure 10 does not teach the features of claim 2.

Additionally, the Examiner cites to the following portions of *Crow* as disclosing the features of claim 2:

> [0038] FIGS. 8A and 8B show how the operating system uses indirect extents to grow the middle of a file. FIG. 8A shows an inode 110 assigned to the file. The inode 110 has consecutive direct extents 111, 113, 119 that point to data blocks 114, 215, 330 storing originally consecutive segments of the file. FIG. 8B shows the final file in which an indirect extent 112 has been inserted between the two original direct extents 111, 119. The indirect extent 112 points to more extents stored in a data block 116. These extents, in turn, point to **new** data block 117 and original data block 215. Since the indirect extent 112 is physically located between the two original extents 111, 119, the segments stored in the blocks 117, 215 (indirectly pointed to) are logically located between the original segments stored in the blocks 114, 330. Inserting the indirect extent 112 has grown the middle of the associated file by logically inserting the segment in **new** data block 117 between the originally consecutive segments in data blocks 114 and 215.

Crow, p. 3, ¶ 38 (emphasis added).

> [0039] The file system, illustrated in FIGS. 5-8B, allows any extent of an inode to be indirect, because the flag field indicates the type of each extent. This free placement of indirect extents within the inodes enables an operating system to logically insert a new data segment between any two selected data segments of a file without physically moving data blocks. To insert a new data segment, the system inserts an indirect extent into the file's inode between the two extents for the selected data segments. Then,

> the system makes the indirect extent point to a data block storing new direct extents that point, in turn, to the consecutive pieces of new data segment. The new direct extents are logically located in the inode at the point where the new indirect extent has been inserted.

Crow, p. 3, ¶ 39 (emphasis added).

> [0042] If the inode has an empty row, the operating system shifts down the original extents corresponding to segments that will follow the segments to be inserted by one row in the inode (step 134). Then the operating system inserts a new direct extent in the newly emptied row of the inode (step 136). Finally, the operating system writes the new file segment to a **new** data block pointed to by the new direct extent (step 138).

Crow, p. 3, ¶ 42 (emphasis added).

> [0044] Next, the operating system inserts an indirect extent into the row of the inode previously occupied by the extent now in the second row of the indirect block (step 146). The new indirect extent points to the new indirect block and has a length equal to the sum of the lengths of both extents in the indirect block. In FIG. 8B, the operating system writes the extent 112 pointing to the data block 116 to the inode 110. Finally, the operating system writes the new file segment in the **new** data block pointed to by the new direct extent (step 148). In FIG. 8B, the new file segment is written to the data block 117.

Crow, p. 3, ¶ 44 (emphasis added).

None of the portions cited by the Examiner or any other portion of *Crow* teaches the feature of, "storing *the additional* data in a *partially filled* block *of another file,*" as recited in claim 2. Paragraphs 38 and 39 describe figures 8A and 8B. As previously discussed, this portion of *Crow* explains how to add ***new*** data blocks to a file using indirect extents. Similarly, paragraphs 42 and 44 describe figure 8C. Both paragraphs specifically state: "the operating system writes the new file segment in the ***new*** data block pointed to by the new direct extent." Therefore, none of the portions cited by the Examiner or any other portion of *Crow* teaches the feature of, "storing *the additional* data in a *partially filled* block *of another file,*" as recited in claim 2. Accordingly, *Crow* does not anticipate claim 2 or any other claim in this grouping of claims.

A.2.ii. ***Rebuttal of the Examiner's Response***

In response to the above facts, the Examiner states that:

> The Examiner respectfully disagreed with the Applicant's argument above; since Crow discloses (paragraph [0036]) "a first portion of the flag field indicates whether the data blocks are locked or unlocked, that is, available or unavailable. The locked designation indicates that access to the data blocks is limited. The processors 44-45 and drivers 47-49 may change the flag field of an extent to the locked designation while manipulating data in the associated data blocks so that other devices do not access the data blocks in parallel. A second portion of the flag field indicates whether empty data blocks have been zeroed. By using the not zeroed designation, the file system can allocate a data block to a file without zeroing the block beforehand. If a subsequent access writes the entire data block, the block will not have to be zeroed saving processing time. A third portion of the flag field categorizes the data type stored in a data block into one of three types, that is, real file data, non-data, or extents".

Final Office Action dated October 18, 2006, p. 6.

The Examiner accurately quotes paragraph 36 of *Crow*, but misinterprets the meaning of paragraph 36 *vis-à-vis* the features of claim 2. The relevant portion of the above-quoted text is as follows:

> A second portion of the flag field indicates whether ***empty data blocks*** *have been zeroed.* By using the not zeroed designation, the file system can allocate a data block to a file without zeroing the block beforehand.

Crow, selected portion of paragraph 36 (emphasis supplied).

Thus, crow states that the second flag field of an inode extent indicates whether an ***empty*** data block has been zeroed. Thus, the data block is already empty, regardless of whether the data block has been zeroed. The term zeroed therefore applies to a concept *other than whether the data block is partially filled*, as required by claim 2. *Crow* explicitly provides that regardless of whether the data block is "zeroed," the data block is empty. Thus, *Crow* does not teach the claimed feature of, "responsive to the additional data being present, storing the additional data in a partially filled block of another file," as in claim 2. Accordingly, *Crow* does not anticipate claim 2 or any other claim in this grouping of claims.

A.3. Claims 4, 12, and 19

Claim 4 is a representative claim of this grouping of claims. Claim 4 is as follows:

> 4. The method of claim 3, wherein the partially filled block is a last block of the another file.

The Examiner rejects claim 4 as anticipated by *Crow*. Regarding claim 4, the Examiner states that:

> Crow discloses in figure 8C, wherein the partially filled block being a last block of the another file (paragraph [0042]).

Final Office Action dated October 18, 2006, p. 3.

The portion of *Crow* cited by the Examiner is as follows:

> [0042] If the inode has an empty row, the operating system shifts down the original extents corresponding to segments that will follow the segments to be inserted by one row in the inode (step 134). Then the operating system inserts a new direct extent in the newly emptied row of the inode (step 136). Finally, the operating system writes the new file segment to a new data block pointed to by the new direct extent (step 138).

Crow, paragraph 0042.

The cited portion of *Crow* teaches that if the inode has an empty row, the operating system shifts original extents in the inode by one row. The operating system then inserts a new direct extent into the newly emptied row of the inode. The operating system then writes the new file segment to a new data block. (Applicants note that the file segment is written to the data block to which the extent points, and not to the inode itself).

However, *Crow* in no way teaches that, "the partially filled block is a *last* block of the another file." In fact, *Crow* does not teach partially filled blocks at all, so *Crow* cannot teach this claimed feature. Certainly, *Crow* does not mention that a partially filled block is a *last* block of the another file, as claimed. Accordingly, *Crow* does not anticipate claim 4 or any other claim in this grouping of claims.

A.4. Claims 5, 13, and 20

Claim 5 is a representative claim of this grouping of claims. Claim 5 is as follows:

> 5. The method of claim 1, wherein the space is located in an extension area in the inode.

The Examiner rejects claim 4 as anticipated by *Crow*. Regarding claim 5, the Examiner states that:

> Crow discloses in figures 5-10, wherein the space being located in an extension area in the inode.

Final Office Action dated October 18, 2006, p. 3.

None of Figures 5-10 teach the claimed feature that, "the space is located in an extension area in the inode." The Examiner does not point to any particular part of these figures as teaching this claimed feature. Instead of showing that each individual figure fails to show the features of claim 4, Applicants use Figure 7 of *Crow* to prove that the opposite is true; namely, that *figures in Crow specifically do not teach* that the space is located in an extension area in the inode, as claimed. A similar analysis applies to each of Figures 5, 6, and 8-10. Figure 7 of *Crow* is as follows:

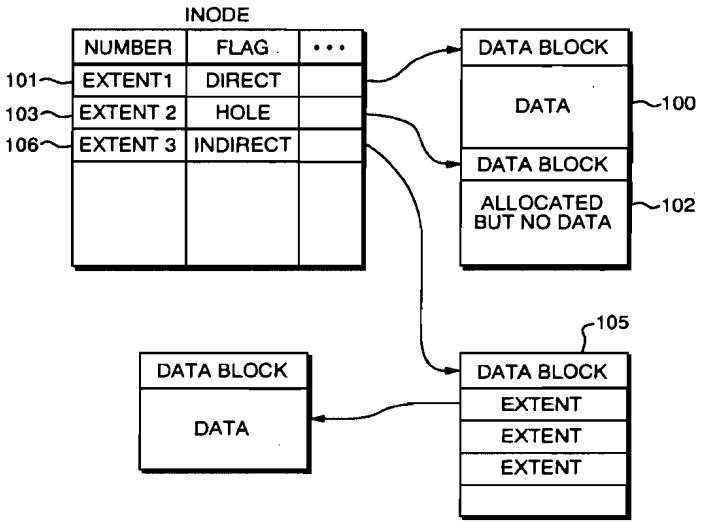

FIG. 7

Figure 7 of *Crow* shows that extents in an inode *point to* data blocks. The *data blocks* contain the stored data. The inode extents do not store the data, only the data blocks. Therefore,

these figures do not teach the claimed feature of, "wherein the space is located in an *extension area* in the inode." Accordingly, *Crow* does not anticipate claim 5 or any other claim in this grouping of claims.

The portion of *Crow* closest to the features of claim 5 is as follows:

> [0053] The operating system writes the binary value to the third entry 176 to indicate storage of a data file when the associated inode is first created. Then, the operating system uses the inode to store the associated data file. When the size of the data file surpasses the limited space available in the inode, the operating system converts the inode to an inode for storage of lists of extents.

Crow, paragraph 53.

This portion of crow states that the operating system uses the inode to store the associated data file. When the size of the data file surpasses the space in the inode, the operating system converts the inode to an inode for storage of lists of extents.

However, again *Crow* does not teach that the data is stored in the *extents themselves*, as required by claim 5. Therefore, this portion of *Crow* does not teach all of the features of claim 5. Moreover, no portion of *Crow* teaches these claimed features. Accordingly, *Crow* does not anticipate claim 5 or any other claim in this grouping of claims.

A.5. Claims 6, 14, and 21

Claim 6 is a representative claim of this grouping of claims. Claim 6 is as follows:

> 6. The method of claim 1 further comprising:
> determining whether a file size for the data is divisible by a block size for blocks in the file system; and
> if the file size is divisible by the block size, storing the data in a block.

The Examiner rejects claim 4 as anticipated by *Crow*. Regarding claim 6, the Examiner states that:

> Crow discloses further comprising determining whether a file size for the data being divisible by a block size for blocks in the file system; and if the file size is divisible by the block size, storing the data in a block (paragraph [0031], [0034]).

Final Office Action dated October 18, 2006, p. 3.

The first portion of *Crow* cited by the Examiner is as follows:

> [0031] Each data block 80-82, 84-85, 92-94 has the same size, for example, 4K bytes. Nevertheless, the extents 65-66 can map file segments of different sizes to physical storage locations. To handle file segments of different sizes, each extent has a length field that indicates the number of data blocks in the string of data blocks that stores the associated file segment.

Crow, paragraph 0031.

This portion of *Crow* teaches that data blocks have the same size. Extents in an inode can map file segments of different sizes to different physical storage locations. An extent length field in the inode indicates the number of data blocks in the string of data blocks that stores the associated file segment.

However, *Crow* does not teach *determining* whether a file size is divisible by a block size for blocks in the file system. *Crow* does not actually store data in a block ***if*** the file size is divisible by the block size, as claimed. Instead, *Crow* only teaches that large files are stored by appending small data blocks. Because this feature is not equivalent to the claimed feature, this portion of *Crow* does not teach the features of claim 6.

Nevertheless, the Examiner quotes from a second portion of *Crow* as teaching the features of claim 6. The second portion of *Crow* cited by the Examiner is as follows:

> [0034] The address pointer field indicates both a logical volume and a physical offset of a data block in the logical volume. In one embodiment, the pointer fields for the logical volume and the data block therein are 2 bytes and 4 bytes long, respectively. For this field size and data blocks of 32 kilobytes, the extent fields can identify about 140.times.10.sup.12 bytes of data in each of about 64K different logical volumes. Thus, the file system of the distributed storage system 40 can handle very large files.

Crow, paragraph 0034.

This portion of *Crow* teaches that the address pointer field indicates both a logical volume and a physical offset of a data block in the logical volume. *Crow* also describes how the described system can handle very large files using small data blocks. However, again, *Crow* does not teach *determining* whether a file size is divisible by a block size for blocks in the file system. *Crow* does

not actually store data in a block ***if*** the file size is divisible by the block size, as claimed. Instead, *Crow* only teaches that large files are stored by appending small data blocks among different logical and physical volumes. Because this feature is not equivalent to the claimed feature, this portion of *Crow* does not teach the features of claim 6.

Neither of the portions of *Crow* teach all of the features of claim 6. Therefore, *Crow* does not anticipate claim 6 or any other claim in this grouping of claims.

B. CONCLUSION

As shown above, *Crow* does not anticipate any of the claims. Therefore, Applicants request that the Board of Patent Appeals and Interferences reverse the rejections. Additionally, Applicants request that the Board direct the Examiner to allow the claims.

/Theodore D. Fay III/
Theodore D. Fay III
Reg. No. 48,504
YEE & ASSOCIATES, P.C.
PO Box 802333
Dallas, TX 75380
(972) 385-8777

CLAIMS APPENDIX

The text of the claims involved in the appeal is as follows:

1. A method in a data processing system for storing data in a file system, the method comprising:

determining whether space is available in an inode for a file in the file system; and

responsive to space being available, storing the data in the inode.

2. The method of claim 1 further comprising:

determining whether additional data is present; and

responsive to the additional data being present, storing the additional data in a partially filled block of another file.

3. The method of claim 1 further comprising:

responsive to spacing being unavailable, storing the additional data in a partially filled block of another file.

4. The method of claim 3, wherein the partially filled block is a last block of the another file.

5. The method of claim 1, wherein the space is located in an extension area in the inode.

6. The method of claim 1 further comprising:

determining whether a file size for the data is divisible by a block size for blocks in the file system; and

if the file size is divisible by the block size, storing the data in a block.

7. The method of claim 1 further comprising:

determining whether space is available in the inode to store the data; and

responsive to room being unavailable in the inode, storing the data in a partially filled block of another file.

8. A data processing system for storing data in a file system, the data processing system comprising:

a bus system;

a communications unit connected to the bus system;

a memory connected to the bus system, wherein the memory includes a set of instructions; and

a processing unit connected to the bus system, wherein the processing unit executes the set of instructions to determine whether space is available in an inode of the file in the file system; and store the data in the inode in response to space being available.

9. A data processing system for storing data in a file system, the data processing system comprising:

determining means for determining whether space is available in an inode of the file in the file system; and

storing means, responsive to space being available, for storing the data in the inode.

10. The data processing system of claim 9, wherein the determining means is a first determining means and the storing means is a first storing means and further comprising:

second determining means for determining whether additional data is present; and

second storing means, responsive to the additional data being present, for storing the additional data in a partially filled block of another file.

11. The data processing system of claim 9, wherein the storing means is a first storing means and further comprising:

second storing means, responsive to spacing being unavailable, for storing the additional data in a partially filled block of another file.

12. The data processing system of claim 11, wherein the partially filled block is a last block of another file.

13. The data processing system of claim 9, wherein the space is located in an extension area in the inode.

14. The data processing system of claim 9, wherein the determining means is a first determining means and the storing means is a first storing means and further comprising:

second determining means for determining whether a file size for the data is divisible by a block size for blocks in the file system; and

second storing means, if the file size is divisible the by the block size, for storing the data in a block.

15. The data processing system of claim 9, wherein the determining means is a first determining means and the storing means is a first storing means and further comprising:

second determining means for determining whether space is available in the inode to store the data; and

second storing means, responsive to room being unavailable in the inode, for storing the data in a partially filled block of another file.

16. A computer program product in a computer readable medium for storing data in a file system, the computer program product comprising:

first instructions for determining whether space is available in an inode of the file in the file system; and

second instructions, responsive to space being available, for storing the data in the inode.

17. The computer program product of claim 16 further comprising:

third instructions for determining whether additional data is present; and

fourth instructions, responsive to the additional data being present, for storing the additional data in a partially filled block of another file.

18. The computer program product of claim 16 further comprising:

third instructions, responsive to spacing being unavailable, for storing the additional data in a partially filled block of another file.

19. The computer program product of claim 18, wherein the partially filled block is a last block of another file.

20. The computer program product of claim 16, wherein the space is located in an extension area in the inode.

21. The computer program product of claim 16 further comprising:

third instructions for determining whether a file size for the data is divisible by a block size for blocks in the file system; and

fourth instructions, if the file size is divisible the by the block size, for storing the data in a block.

22. The computer program product of claim 16 further comprising:

third instructions for determining whether space is available in the inode to store the data; and

fourth instructions, responsive to room being unavailable in the inode, for storing the data in a partially filled block of another file.

EVIDENCE APPENDIX

There is no evidence to be presented.

RELATED PROCEEDINGS APPENDIX

There are no related proceedings.